Adaptive Waypoints for Graduated Embodiment of Autonomous Mobile Robots

Mark Boslough
Evolutionary Computing & Agent Based Modeling Dept.
Sandia National Laboratories
Albuquerque, NM 87122
mbboslo@sandia.gov

Arthurine Breckenridge
Intelligent Systems Principles Dept.
Sandia National Laboratories
Albuquerque, NM 87122
arbreck@sandia.gov

Michael Peters
Evolutionary Computing & Agent Based Modeling Dept.
Sandia National Laboratories
Albuquerque, NM 87122
mpeters@sandia.gov

Abstract

We describe the methods we are developing to design behaviors for embodied agents in the form of autonomous vehicles. Our work is based on a strategy we call *graduated embodiment*, in which high-level behavior algorithms that are initially developed using evolutionary computing methods in a relatively low-fidelity, disembodied modeling environment can be migrated to useful applications. We argue for biomimetic behavior engineering that is a hybrid of human design and artificial evolution, and evolutionary computing that is applied in stages to preserve building blocks and limit search space. We present our application of these methods to the concept of *adaptive waypoints*, which allow navigation behaviors to be re-used among vehicles with different degrees of embodiment, levels of fidelity, and modes of locomotion.

1. Introduction

The tasks of trying to figure out where you are, where you are going, and how to get there are some of life's oldest dilemmas. Navigation, positioning, and path planning are crucial to virtually every activity undertaken by animals and humans, yet the process often seems impossible to describe, model, or prescribe. In the simplest examples, the goals and rules of motion allow for simple behavior algorithms that can be turned into human-generated computer programs that can control human-created systems. But such programs are brittle and prone to failure if the system does not behave exactly as anticipated.

The real world is full of contingencies, unexpected events, multiple (and often competing) goals, nonlinear responses, feedbacks, noise, and complex interactions across a wide range of time scales and levels of organization. Nevertheless, animals have evolved the ability to cope, prosper, and multiply in such a world. Their robust behaviors give them the ability to navigate and "plan" their path of motion in order to migrate, search for food, return home, find mates, and avoid predators. They are successful at these goals while simultaneously tending to lower-level tasks, adapting to changes in their environment, dealing with unfamiliar situations, and ignoring irrelevant information.

Human-designed autonomous mobile robots must possess similarly complex and adaptive behaviors if they are to be useful for the types of problems for which they are being proposed. Many of their goals are analogous to those of animals, and include migration, search, cooperation, exploration, location, and avoidance. In some cases, the adaptive behaviors of animals can be applied to the development of autonomous robots; a method we call "biomimetic behavior engineering" (Boslough, 2002). Locomotion behaviors (such as dynamic soaring by albatrosses) can be analyzed and written into a control algorithm, but such purely "hand-coded" behaviors are usually unable to deal with the unexpected.

The simple borrowing of behaviors observed in the animal world and applying them to robotics is insufficient. The external manifestation of a behavior can be simulated, but the actual behavior includes the internal processing. Real animal behaviors tend to be bottom-up and emergent, not top-down and "hard-wired". One can write a computer program that will *appear* to code a behavior that, in reality, emerges from a complex nonlinear dynamic balance of different processes. Such programs can be useful for applications where the environment is predictable, but they amount to mimicry of external expression, not internal process. The "external" method of biomimetic behavior engineering is a form of "classical artificial intelligence (AI)" and suffers from the same limitations.

Our goal is to develop a paradigm for behavior engineering that generates robust, re-usable, and useful control systems for autonomous mobile robots. Our primary application of interest is to develop goal seeking navigation behaviors.

2. Previous Research

As this work builds on the work of others the related literature is presented with an overview of genetic programming and autonomous mobile robots, with an emphasis on navigation.

2.1. Genetic Programming

Research in the planning and control of mobile robots has received much attention in the past two decades. We have chosen genetic programming (GP) methods to develop our robotic behaviors because this method has been demonstrated to work for "proof of principle" problems. In his book, *Genetic Programming III*, Koza (1999) documents sixteen attributes that are needed for challenging a computer to solve a problem without explicitly programming it. No other methods come as close as GP, which currently unconditionally possesses thirteen of the sixteen attributes. GP starts with a high level statement of what needs to be done, determines a basic sequence of how to do it, produces a computer program, automatically determines the size of the program, provides code reuse, provides parameterized reuse, determines internal storage needs, determines iterations, loops, and recursions, organizes into hierarchies, automatically determines architecture, implements a wide range of programming constraints, and operates in a well-defined way.

We describe an approach where the GP at least partially possesses the last three attributes: wide applicability, scalability, and competitiveness with human-produced results. The attributes are needed to produce the ultimate goal of the system for automatically creating computer programs to produce *useful* programs.

We extend the work of Pryor (1998) and Barnette *et al*. (2000) who applied evolutionary methods to the development of robotic behavior. Pryor (1998) originally developed a genetic programming model to solve a suite of high-level robotics problems. The motion of Pryor's "robugs" was idealized and highly constrained so that the focus of his evolutionary model could be on high-level navigational behaviors and goals. Low-level maneuvering issue--such as locomotion, steering, and braking control--were not initially addressed. Instead, the simulated robugs were constrained to move on a discrete two-dimensional square grid with instructions such as "turn" or "move ahead".

Behaviors were automatically generated using a genetic program to solve a series of problems in which robots are randomly distributed on a grid with obstacles and rewarded for finding a source that is emitting a signal. The required high-level navigational behavior can be encapsulated as a computer program, which can be engineered by a variety of methods. In the simplest case—when effective rules are easy to conceive and implement—the behavior can be coded by hand. For trivial goals, common sense is all that is required for inventing rules. More difficult challenges require more sophisticated solutions, which can be generated by a variety of optimization methods including thermodynamic analogy models, conventional guidance theory, or reinforcement learning.

Pryor (1998), and Barnette *et al*. (2000) outline the details of the implementation, and the specifics are not repeated here. The representation allows a great deal of flexibility, and can be adapted to many types of problems. The behavior programs execute by traversing a tree that is made up of building blocks called nodes, which can either be a function or a terminal. Functions perform operations and contain pointers to other nodes. Terminals return values that result in an instruction to the robot. The trees themselves are generated by a genetic programming model originally developed by Pryor (1998) and based on methods described by Koza (1992), within a general framework presented by Holland (1975). Genetic programming is a type of genetic algorithm, an evolutionary computing method that is based on the principles of biological evolution.

Artificial evolution takes place over many discrete steps called generations. Generations in nature are not synchronous because lifetimes and breeding times vary in length, but evolution has been in operation for billions of years on Earth. In the model they are synchronized for simplicity, and the number of generations is limited by practical considerations to hundreds. Each generation consists of a population of individuals. In nature these are organisms and the population size can vary and can reach numbers of millions or billions. In the model they are computer programs and the populations are held fixed for a given problem, with typical sizes on the order of thousands. The Darwinian principle of "survival of the fittest" is applied. In nature, any individual that survives long enough to breed and generate offspring is fit by definition. In the model, each individual behavior program is assigned a numeric score based on a "fitness function" chosen by the researcher to represent the problem, and survival depends on rank. Finally, the individuals must reproduce. In nature, the genotype is carried by DNA, which is mixed between individuals by sexual reproduction, and random processes occasionally introduce mutations. The human genome contains between 30,000 and 100,000 genes representing many millions of base pairs. In the model, the "genotypes" of two individuals are mixed with a crossover operator, and random mutations are introduced to allow exploration of other regions of the "fitness landscape" (see Koza (1992) for more a more detailed descriptions of these concepts). These artificial genomes have much lower information content than DNA in nature, containing hundreds of genes represented by thousands of bits of information.

2.2. Autonomous Mobile Robots

An autonomous mobile robot in this work refers to software construct situated in an environment in which it is capable of self-directing freedom in pursuit of its goals. (Wiess, 1999) An agent is embodied if it occupies a space in an environment and affects the environment. Embodied agents consists of a) sensors used to observe their worlds, b) the cognitive component used to plan an action, and c) actuators (or effectors) used to carry out the action.

In a deliberative, logic-based paradigm, mobile robot control is divided into sense, plan, and act phases, with each phase handled respectively (Gat, 1997). The cycle is repeated at regular and very small time intervals throughout the agent's operational period. This paradigm works best for well-defined, representable problems. Unfortunately, many of the environments where embodied agents might be used (certainly in most real world environments) demand that the agents make decisions with alacrity due to the high rate of environmental change. As environments become more and more dynamic and complex, the problems become more severe, eventually causing the agents based on a sense/plan/act cycle to become impractical (Brooks, 1991).

In an effort to solve these very difficult problems facing autonomous systems, the new approach given many names by different researches, including the "animat" approach (Wilson, 1985), task-oriented subsumption architecture (Brooks, 1986), computational neuroethology (Cliff, 1991), and behavior-based AI (Maes, 1993) has emerged. Rather than explicitly planning, the reactive, behavior-based agents would simply re-observe the world at every computational step and act on what they perceive at that instant. The agent's decision-making function is assembled from a number of simple behaviors that are triggered in response to sensory input. The simple priority-based behavior arbitration and lack of complex planning allow agents to react quickly to situations that demand it. However, while solving one problem by being adept at reacting to situations and real-time decisions, several others--such as scalability, inability to learn from past mistakes, and lack of cooperative behaviors--were introduced.

In Wooldridge & Jennings (1994) previous research in these approaches, as well as very active research in hybrid architectures, have been summarized. Their strengths are complementary and combining the two approaches can mitigate some of their weaknesses. In a layered, hybrid approach, the agent's subsystems are arranged in a hierarchy with higher layers dealing with information at increasing layers of abstraction.

We describe a hybrid approach that addresses the challenge of achieving an optimal (or least acceptable) balance between reactive and deliberative behaviors. It differs from most existing hybrid architectures in two key ways. First, it uses a combination of fidelity levels from each approach. Second, it makes use of a vertical and horizontal hierarchical structure to manage the interaction between the layers.

2.3. Navigation

Map-learning and path-planning strategies are very active research in the context of autonomous vehicles. Current literature (Meyer & Filliat, 2003; Filliat & Meyer, 2003) discusses numerous navigation strategies and outline the adaptive capacities each afford to cope with complex, dynamic environments. The algorithms insist on being tightly coupled with the vehicle that is used.

As a benchmark for implementing and testing our various concepts, we are using a grid-based UAV (or bird) problem defined by Pryor & Barton (2002). The problem is similar to the robug problem in that the bird is constrained to move on a two dimensional Cartesian grid. It requires flight behavior logic to search for uplift regions and investigate the surrounding area without crashing. This problem is more complicated than the robug problem because of the different procedures that must be done sequentially. The agent must search and map the uplift region before the surrounding area can be explored, and conflicting goals of exploration versus exploitation (of the lift zone) must be balanced. This conflict was too difficult for a conventional genetic program, and more advanced methods had to be added.

A rectangular "lift zone" is generated with its short dimension fixed at 20 units and its long dimension randomly chosen from a range of between 85 and 105 units. The long-axis is randomly oriented along the *x*, *-x*, *y*, or *-y* axis of the Cartesian coordinate system. The rectangle is placed such that the origin is 5 units from one end, and centered on its width. This leaves between 80 and 100 units extending on one of the four directions. A bird is placed with its orientation chosen randomly from one of the four directions at a random location within a 100×100 square box centered on the origin, at an altitude of $|x| + |y| + 20$ units. When it is outside the lift zone, it loses one unit of elevation for every time step. Inside the lift zone it gains one unit, up to a maximum of 250. It moves one positive or negative unit along either the *x* or *y*-axis, every time step. It can go straight, turn left, or turn right (a modification to the benchmark allows for U-turns). The fitness is defined by the maximum distance away from the origin that a bird reaches before returning to the lift zone. The actual fitness is associated with the evolved behavior program, not with a given instantiation of the bird, so the fitnesses of individual birds running the same program--but with a range of initializations—are averaged. If its altitude goes to zero, a bird crashes and dies, and its contribution to the fitness function is assigned a large negative value.

This would appear to be an extremely simple problem compared to actual flight, but it involves staged and somewhat contradictory goals that force a balance between

exploration and exploitation. The conservative bird will stay in the lift zone to preserve its life, but will get a low score. The bold bird will fly away from the lift zone and increase its risk of crashing. The highly ranked bird will require a delicately tuned algorithm. The benchmark also involves a set of goals that must be accomplished in sequence: 1) the bird must find the lift zone, 2) the bird must exploit the lift zone, 3) the bird must explore away from the lift zone, and 4) the bird must return to the lift zone.

To tackle problems with this level of complexity, we are now developing new methods of adaptive waypoint following as a means of high-level guidance to vehicles without regard to lower-level propulsion and physics or middle-level autopilot algorithms, described in Section 6.

3. Evolution, Design, and Embodiment

Embodiment is the ability of a system to adapt to, learn from, and develop with its environment. Embodiment determines whether that system will "survive" in the environment. More elaborate autonomous mobile robots require increasing computational effort that often proves too cumbersome and slow for real-world applications. The term, *graduated embodiment*, is the notion of mobile robot adapting to appropriate fidelity ranges. A physically embodied agent must have:

- The ability to coordinate actuator and sensor modalities to explore its environment,
- Goal-oriented behavior on micro and macro levels,
- Bi-directional interaction between the agent and its environment,
- Bi-directional communication between the agent and other agents in the environment, and
- An understanding of the physics of the environment. (Duffy, 2001).

Biological evolution itself followed a path of graduated embodiment. Building blocks for biochemical pathways and structures developed during the billions of years before life was fully embodied, when it consisted of single-celled prokaryotes with limited sensory connections to a relatively stable, uniform, and predictable environment. Most of evolutionary history was spent developing the tools that would later be combined and recombined while graduating through higher levels of embodiment, from the first multicellular animals (metazoa), through *homo sapiens*.

Animat research attempts to follow closely with the thought that animals are created by biological evolution, not by a designer. Animals thrive under real-world conditions because their forms and behaviors are honed by billions of years of natural selection. Our modern understanding of the origin animal behavior has its roots in Darwin's four basic postulates:

- Variation exists among individuals in a population.
- Some of these variations are inherited by the next generation.
- More offspring are produced in every generation than are able to survive.
- Individuals with the most favorable variations are the ones that survive and reproduce.

Consequently, survival and reproduction are not random. Favorable variations are selected by nature. Forms and behaviors that appear to have been designed are actually the result of Darwinian evolution.

Evolutionary approaches to behavioral engineering are biomimetic at deeper level, because they mimic the adaptation that led to the structures and behaviors themselves. Evolutionary methods are inherently suitable for application to the graduated embodiment strategy, in which high-level behavior algorithms that are initially developed using evolutionary computing methods in a relatively low-fidelity, disembodied modeling environment can be migrated to useful applications.

Evolutionary computing is appropriate for behavior engineering because it works with behavior building blocks that can be re-used and re-combined, much in the same way that biology operates. In biological systems, information is encoded in DNA, the raw material on which evolution operates. DNA holds the internal representation of an organism, its genotype. The outward manifestation of an animal, its phenotype, includes the reflexes, behavior, and intelligence that allow it to survive and reproduce as an embodied entity. "Internal" biomimetic behavior engineering emulates life in this respect, and generates behaviors that emerge by operating directly on the genotype—in this case represented by computer code. It depends on some degree of embodiment, and is a form of "new AI".

A drawback of artificial evolution is that it does not come close to the fidelity of evolution in the natural world in terms of numbers of generations, number of individuals in a population, or information content of the genome. However, the largest obstacle seems to be the fact that evolutionary computing methods are simulations that model "virtually embodied" entities, whereas natural evolution operates in the real world on actual embodied individuals. Unless the "off-line" simulation environment captures the important characteristics of the real world, successful behaviors may not survive when exposed to reality. In biology, on the other hand, the fitness of individuals has always been tested "on-line" under actual survival conditions.

It is not a profound observation that virtual embodiment is not the same as actual embodiment. The best argument for the physical grounding hypotheses of new AI is summed up by Brook's (1990) remark that "…the world is its own best model. It is always exactly up to date. It always contains every detail there is to be known." But to evolve robotic

behaviors from scratch in the real world would be impossible for lack of prokaryotic, self-replicating robots and billions of years. This is an extreme restatement of the fact that strong embodiment is not possible with current technology (Sharkey & Ziemke, 2000). A realistic strategy for artificial evolution is to skip the impossible steps and graduate from virtual to physical embodiment through increasing levels of fidelity.

Natural evolution's embodiment graduated from simple to complex organisms. Only during the most recent ticks of the geologic clock did biological creatures possess the directed mobility, good eyeballs, and large brains that appear to be prerequisites for truly intelligent behavior (e.g. Moravec, 1984). Technology already permits robots with these physical attributes, but they seem to be insufficient because of the lack of *history* in the software. Biological evolution is a historical process that takes place in a world where the future cannot be predicted. The DNA that encodes animal behaviors contains information built up over the entire span of geologic time. Successful behaviors are the ones that contributed to survival and reproduction throughout countless generations, during times when the world may have been very different. The value of a particular behavior depends on how closely the present resembles the past. The ability of behavioral adaptations to emerge in response to changes in the environment depends upon rich and useful genetic variability, which results in part from history.

In the natural world, evolution simultaneously operates on an organism's morphology and behavior; there is no qualitative distinction between these two types of adaptations. This is the approach taken by Sims (1994) whose virtual animats involve co-evolution of both form and function. This strategy greatly expands the solution space that can be explored and allows the animat morphology itself to be part of the optimization process. Sims' animats are, however still subject to limited degrees of freedom and constraints that are not imposed in the natural world. Moreover, the optimal solutions do not necessarily yield physical systems that could lead to an engineering design that would allow embodiment to be realized in the real world.

Engineering considerations require that hardware and software design be decoupled in order to achieve graduated embodiment. The alternative would be to redesign new hardware for every generation (with appropriate genetic variability) to reach an optimum. A more realistic approach is to develop building-block behaviors that can be applied across a broad range of hardware designs. As the physical robot design changes, a new behavior can be allowed to evolve from the pre-evolved building blocks. This method is a hybrid of engineered physical attributes and evolved behaviors, but takes advantage of "history" information embedded in the artificial genome that was accumulated during lower levels of embodiment and fidelity.

Some practitioners of evolutionary computing adopt a purist philosophy that "evolution should be allowed to operate on its own and eventually it will find the best solution." This hands-off approach attempts to bypass any human intervention whatsoever, presumably because human biases might prevent global best--but unanticipated--solutions from being discovered by machine. We reject this approach. It is impossible to carry out for two reasons. First is scalability; unlike the natural world, artificial evolution does not have access to billions of years, the high information density of DNA, nor the large populations of individuals that can harbor a wide range of genetic variability. Second is the fact that artificial evolution by computer can never be free of human biases, because evaluation functions used to rank the success of various phenotypes are written by humans and already contain biases.

Moreover, for robotics the hardware is also human designed. The problem—by its very definition—is a hybrid that contains both designed and evolved components. There is already an interface between designed hardware and evolved software, and this interface must include designed software components--such as device drivers for sensors and actuators--to function. The boundary between designed and evolved components is arbitrary. For all these reasons, we have adopted what we view as the most pragmatic approach in which genetic programming methods are used to extend, but not replace, the creativity and intelligence of the designer. There is no such thing as cheating by hand-writing useful functions, but ultimately it the "natural selection" that decides when to use designed building blocks, and when to use those that have been evolved from scratch. The result is a hybrid of designed and evolved components at all levels of organization.

4. Evolution with Building Blocks

The backbone of our research makes use of building blocks in evolutionary modeling and simulation. For engineered software, the concept of building blocks allows various parts of code to be encapsulated according to function. Software that is not built in this way degenerates into "spaghetti code" that is fault-intolerant and difficult to modify or improve. Likewise, if evolutionary methods are used to generate programs to accomplish large and complex sets of tasks, the resulting code is often inflexible and a poorly optimized dead-end.

Building blocks can be combined either horizontally or in a vertical hierarchy. The most effective strategy is to mix evolved building blocks with engineered or hand-written building blocks. For vertically integrated problems, some aspects are easy to build by hand, whereas others are very difficult or impossible when the model has many degrees of freedom. Our effective strategy is to mix evolved building blocks with hard engineered or hand-written building

blocks. Our intent is to evolve functionality requiring human and machine collaboration.

Rattlescape: A simple conceptual model can be used to illustrate these ideas. Suppose, for example, we wanted to design a behavior for a robotic rattlesnake whose only goal is to survive in an environment where it might be stomped by a large grazing animal. The human-designed snake hardware would consist of sensors and actuators. We want a snake that can sense, slither, and rattle like a real snake. A greatly simplified snake animat might have a dozen sensors and a hundred actuators. Evolution from scratch is impossible for reasons of scalability.

We know from human experience that a buffalo will avoid stepping on a buzzing rattlesnake, so we could write a driver by hand that activates the tail vibration actuators in a cycled sequence at a frequency that creates the appropriate sound that will startle a buffalo.

Human observation also tells us that snakes exhibit several different modes of locomotion, most of which involve wavelike motion due to muscles contracting in a cyclic sequence. Armed with this knowledge, we may wish to reduce the number of degrees of freedom by considering, for example, symmetric and anti-symmetric normal modes as the building blocks of locomotion, and allow artificial evolution to operate in a limited virtual world to select the phases and amplitudes that yield the most efficient "slither".

However, we might not know the best way to detect a large mammal, other than to suspect that the task requires some combination of vibration, heat, and visual motion sensing. The large-mammal-detection algorithm might be evolved from much smaller sensor data building blocks, through a sequence of virtual worlds that model the coupling of ground motion to vibration sensors, and heat to infrared sensors.

We now have three algorithms with three different levels of organization, and three different types of connections between their designed and evolved components. If they can be considered independent behaviors, another stage of artificial evolution can be used to find a way to combine them that leads to the best snake survival rates. An obvious solution that would be quickly found by a genetic program would be that shown in Figure 1: near animals would activate the rattle behavior, and distant animals would activate "slither".

Now suppose the world changes. The buffalo are gone and replaced by human beings who react to buzzing rattlesnakes by killing them. The same building blocks can be put together in a different way to generate a new survival behavior. In the successful snake animats, the detection algorithm now inhibits the vibration driver and perhaps activates the slither driver for all detections.

For this illustration we have made an assumption that will often fail in practice. We assumed that the three high-level behaviors operate in complete isolation. It may turn out that slither and rattle use common actuators and cannot be simultaneously activated. Another possibility is that the detection algorithm requires a stationary snake. Treatment of the behaviors as independent building blocks greatly reduces the number of degrees of freedom, but it may also prevent the discovery of new survival adaptations and higher-level optimization.

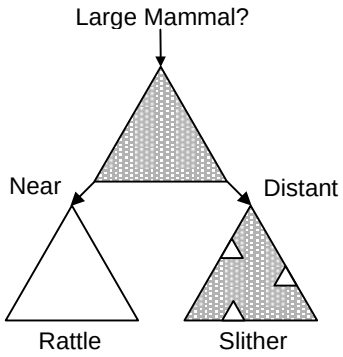

Figure 1: A composite behavior tree is constructed of three subtrees which can be hybrids of sub-subtrees that have both designed (white) and evolved (gray) components at different scales of organization.

Our way of dealing with this is to apply staged optimization, in which various high-level behaviors are evolved independently as in the above example. A global optimization is then performed by holding some behaviors fixed and allowing others to co-evolve with the total system. This might lead to a new emergent behavior, such as "rattle and slither", if that is a positive survival adaptation.

5. Staged Optimization and Pruning

For the same problems, the behavior optimization process is much more efficient when a complex task is broken into smaller subtasks. For the benchmark problem described in Section 2.3, the initial genetic flight control algorithms that have been developed involve mere thousands of operations and can be described by a "genome" of a few thousand bytes. A simple doubling of the number of tasks that the behavior must manage--or a similar increase in fidelity--would put the problem out of reach on most compute clusters. Building block efficiency seems to work best when the various subtasks are independently optimized in parallel by using a "policy table" which can be expressed as a root node with a set of pointers to the various behavior trees.

Many iterations of Pryor's (1998) genetic programming tool were implemented to tackle this problem, with varying degrees of success. Versions of the code were written with single trees, multiple trees, and various trees executed in sequence. In some, a "ponder tree", which was intended to act as a higher processing layer, was inserted between the sense level and the decision trees which output the action to

be taken. In Figure 2, the relative success of various methods can be seen as measured by the maximum fitness value reached by various genetic programs, and the rate at which it is reached. As it turned out, the existence of an evolved ponder tree inhibited the rate of optimization.

We have developed generalized versions of Pryor's GP, with the added goals of 1) platform independence, 2) modularity, and 3) integrated visualization tools. The data structure of the tree that contains the flight control algorithm consists of multiple arrays that keep track of state variables (for flight simulation), calculated variables (for high-level situational awareness and decisions), calculated integers (for policy table flags), registers (genetically calculated values) and pointers (for multiple decision trees that depend on policy flags). Array dimensions are determined at compile time, and depend on the evaluation function, the number of degrees of freedom of the flight vehicle, and whether the flight space is discrete or continuous. By generalizing the genetic programming, behaviors for any environmental rules can be developed with the same code, from the most basic (4 degrees of freedom, discrete space, simple flight rules, fixed lift zone) to the most advanced (6 degrees of freedom, continuous space, full aerodynamics, turbulent boundary layer). We used the most basic example as a benchmark because it runs much faster than a simulation that must solve realistic equations of motion at every time step.

The first version that demonstrated fast convergence and approached the theoretical best solution for the benchmark problem is the one that implemented a policy table. This makes sense because of the sequential and competing goals described earlier. It is because a bird in one situation, such as needing to find the lift zone, needs to be running a very different program than one in a situation where long-distance exploration is paramount. For that reason, several trees were evolved independently, and each were called based on the a policy table value that represented its current situation, such as 1) the bird has not discovered the lift zone, 2) the bird is inside the lift zone, 3) the bird has reached and altitude of 250, or 4) time is running out. When such a policy table was implemented, the ability of the code to find good solutions increased markedly.

Staged optimization (Figure 3) led to even faster improvement in evolved behaviors, but required more human intervention. In the staged process, one or more sub-behaviors can be locked, while the remainder are independently optimized. This procedure greatly reduces the parameter space to be searched by the program. Combining staged optimization with sequential pruning of the behavior trees led to the fastest improvement. In sequentional pruning (Figure 4), the behavior code is reduced in size by allowing the genetic program to remove superfluous parts by applying a size penalty to the fitness function. By locking all trees except the one to be pruned, this process can be accelerated.

Platform independence is important for computationally intensive genetic programming applications. This was achieved by implementing a text-based (and in later versions, an XML-based) description of the program trees in place of the original binary representation. These text trees are compact, with about a 10 Kbytes storage requirement per behavior. Because the computation-to-communication ratio can be extraordinarily high for this application, extra bandwith, or even the latency of writing to disk, is not a major problem.

The code was reorganized so that the evaluation routines were broken into modules that could be called either from the genetic programming tool or another (e.g. visualization) program. By keeping the modules independent, the code could be applied to other problems simply by inserting a different evaluation function. In the most recent version, the

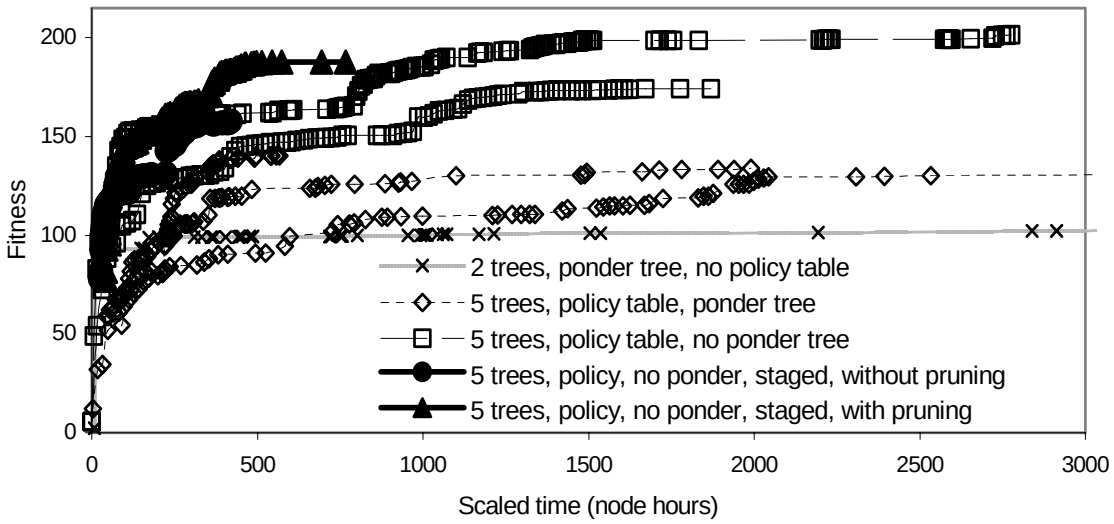

Figure 2: Benchmark problem performance on Sandia's Cplant. The theoretical best fitness value is 215.

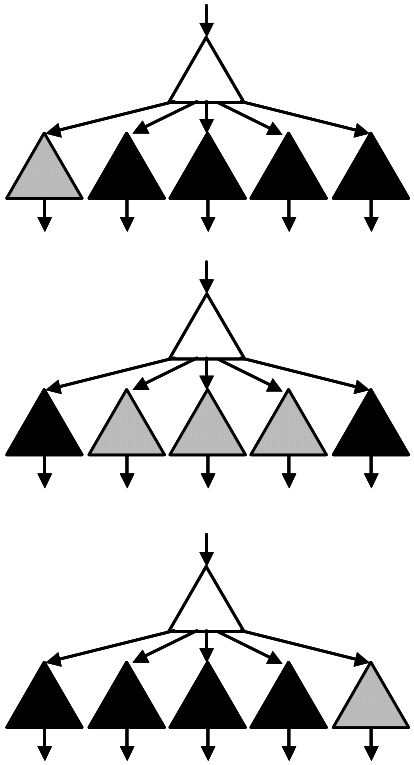

Figure 3: Staged optimization reduces search space. White tree is a designed policy table, black trees are locked, and gray trees are undergoing evolution.

code has been converted to a fully object oriented representation.

As a more complex test problem for evaluating our implementation of building blocks, we merged our benchmark lift zone problem with an image-searching problem. We separately evolved a single-tree image zone search behavior, and then combined it with our five-tree lift zone algorithm to form a new, more complex six-tree behavior. Because we are able to store series of trees as ASCII text files, the merging of behaviors is a simple as file concatenation (although files can also be kept separate). The horizontal integration directly addresses the scalability of GP programs.

We recognize that some complex real-world behaviors will be much more difficult to break apart into smaller units, and in other problems the subtasks are not independent and must be optimized in consort. We think that by using building blocks we can allow the behavior designer to make the decision as to what constitutes a subtask, and to provide flexibility as to which sets of subtasks are most effectively co-evolved.

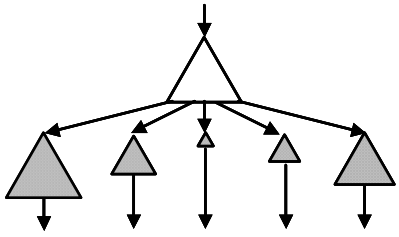

Figure 4: Pruning of trees removes useless code.

6. Navigating with Adaptive Waypoints

Integrating behavior algorithms in a vertical hierarchy is also required to create a comprehensive behavior system that includes cognative, instinctual, and reflexive levels of behavior in addition to physical model attributes. In one example of vertical integration, we are implementing *adaptive waypoints* (Figure 5) to provide a means of high-level guidance to vehicles without regard to lower-level propulsion and physics or middle-level autopilot algorithms. Adaptive waypoints can be used to capture goal-oriented and collective agent behaviors that can be transferred from one type of vehicle to another as a self-contained behavioral "building block". Each individual vehicle is guided by its own dynamic waypoint, which moves in such a way as to lead the vehicle toward its individual or collective goal.

Waypoints are coordinates in a virtual world. An entity can follow waypoints in the order they are given (e.g., read coordinates from a file). Or, entities are often assigned to perform certain tasks (e.g., loiter) once they reach a certain location. Waypoints can be either user-generated or entity-generated, and can be static, dynamic, or adaptive. A waypoint can thought of as an internal representation, or part of an agent's cognitive map.

Waypoints can take on various attributes of longevity, type-identification, and goal-orientation.

Longevity: Static waypoints are intended to be permanent navigational aids and can be stored. Dynamic waypoints are intended to mark locations that are of temporary tactical importance (e.g., the last known location of an enemy). While static waypoints remain in the entity memory queue indefinitely, dynamic waypoints have a clearly defined lifespan. After the lifespan is up or fade factor expires, the dynamic waypoint expires and is no longer stored by the entity. Adaptive waypoints can either be static or dynamic but their longevity is determined reactively since it responds in a timely fashion to changes in the environment. Adaptive

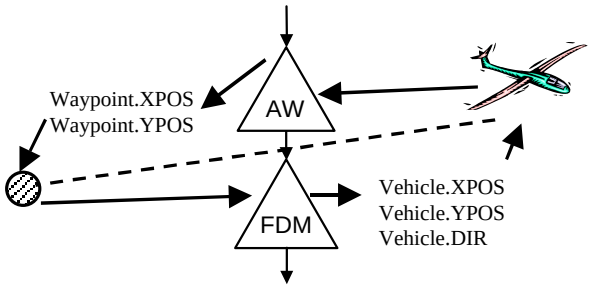

Figure 5: Adaptive Waypoint (AW) and Flight Dynamics Model (FDM).

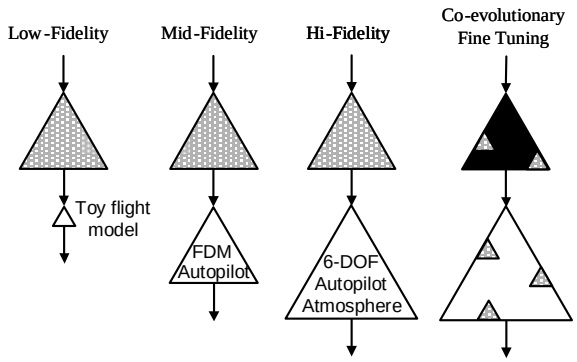

Figure 6: Graduated embodiment of adaptive waypoint takes behavior through stages of evolution.

waypoints are determined by continuously running processes.

Type identification: All types of waypoints can be tagged with types to identify what kind of behavioral information they represent or any other information in addition to position. Static and dynamic waypoints have fixed type identification. Adaptive waypoints are autonomous since they exercise control over their own actions and can take on a new identities when needed. Identity can be learned and changed based on previous experiences.

Goal-orientation: Often, waypoints are used to solve situational goals. Static waypoints have no goal-orientation. Dynamic contain information that may aid others in obtaining a goal. Adaptive waypoints do not simply act in response to the environment but are goal-oriented and can communicate with other agents to determine a goal.

The intelligence of the waypoint can be transferred from one type of vehicle to another as a self-contained behavioral "building block." Each individual vehicle is guided by its own adaptive waypoint, which moves in such a way as to lead the vehicle toward its individual or collective goal. The intelligence of a vehicle can be contained entirely within the adaptive waypoint that is aware of and can respond to the state of the vehicle, the vehicle's environment, it's high-level goal, and (in the case of collective behavior) the state of neighboring vehicles.

At every time step, the vehicle position is updated based on the position of its waypoint, on its state, on its dynamics model, and on the autopilot system whose goal is to take it to the waypoint. The waypoint coordinates are updated based on its behavior algorithm in combination with the location and direction of the vehicle, but not on the details of the internal state of the vehicle.

Adaptive waypoint behavior algorithms can be developed using the methods of genetic programming and graduated embodiment by staged optimization (Figure 6). The advantage of graduated embodiment is that behaviors can be evolved in the absence of a high-fidelity vehicle model, reducing the computational cost. Basic waypoint behaviors can be developed using low-fidelity vehicle models that only approximately reflect the actual vehicle performance. Once a waypoint behavior is developed for the low-fidelity vehicle model, the behavior can be tuned through successive stages of higher fidelity, until the highest fidelity model is achieved. This is the essence of graduated embodiment.

Example problem: We can illustrate these concepts with a stripped-down problem that shows that a simple adaptive waypoint that can be graduated from a one-dimensional agent to a vehicle that moves in two-dimensions. Suppose we have a vehicle (V), a target (T), a fixed waypoint (FW) an adaptive waypoint (AW), and a home (H). Let the initial position of V=0 and FW=100. T is a random number between 1 and 200, and H=0. The position of AW at every time step is entirely determined by the GP.

For the simple world, the rules of motion allow V to take one step in the direction of AW every time step. The rules of detection are that V detects T when they have the same coordinate. To be successful, V has to locate any target between 1 and 100, and return to H. The evaluation function is such that the most fit behavior is the one that completes the goal in the shortest time, averaged over some number of tests.

The behavior can easily be coded by hand, but our GP discovered it within the first few iterations: The AW is set at infinity until the vehicle reaches either T or FW, and then AW is set to negative infinity. When the vehicle is allowed to explore in a two-dimensional plane, with a constraint on its turning radius, but otherwise following the same rules of motion, the adaptive waypoint no longer works because the vehicle does not return home (because of the offset due to its turning radius). A single mutation from the first solution will, however, solve the problem. The final AW location evolves from negative infinity to zero.

The point of this illustration is that in going from the simple to the more complex, many parts of the problem may already be solved and those behaviors can be incrementally evolved in a way that allows building blocks and information to be re-used.

7. Conclusions

We have introduced a hybrid AI approach that has many advantages for generating autonomous mobile robot behaviors that operate in a complex, dynamic environment. This method is needed to generalize across different conditions because of the diversity of physical environments, vehicles, locomotion abilities, and interaction dynamics with other objects, We believe navigation strategies, such as, "search-for-target", "move-to-goal", and "avoid-collision", are reusable using the method of adaptive waypoints, provided they are adjusted to suit the physical capabilities of each vehicle using *graduated embodiment.*

8. Acknowledgements

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000. This project was funded by the Laboratory Directed Research and Development (LDRD) program.

References

Barnett, D.W., Pryor, R.J. and Feddema, J.T. (2000). Development and application of genetic algorithms for Sandia's RATLER robotic vehicles. SAND2000-2846, Sandia National Laboratories.

Boslough, M.B.E. (2002). Autonomous dynamic soaring platform for distributed mobile sensor arrays. SAND2002-1896, Sandia National Laboratories.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation,* RA-2(1):14-23, 1986.

Brooks, R.A. (1990). Elephants Don't Play Chess. *Robotics and Autonomous Systems 6*. 3-15.

Brooks, R.A. (1991). Intelligence without reason. *Proceedings of the 1991 Int. Joint Conference on Artificial Intelligence*. 569-595.

Cliff, D. (1991) Computational neuroethology: A provisional manifesto. *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*. 29-39. MIT Press.

Duffy, B.R., Joue, G. (2001). Embodied Mobile Robots. *1st International Conference on Autonomous Minirobots for Research and Edutainment - AMiRE2001*.

Ferguson, I. A. (1992). Towards an architecture for adaptive, rational, mobile agents *Decentralized AI 3 - Proceedings of the Third European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds (MAAMAW-91),* 249-262. Elsevier Science Publishers.

Gat, E. (1997). On three-layer architectures. *Artificial Intelligence and Mobil Robots.* AAAI Press.

Holland, J.H. (1975). *Adaptation in natural and artificial systems*. Univ. of Michigan Press, Ann Arbor.

Koza, J.R. (1992). *Genetic Programming*. MIT Press.

Koza, J., Bennet, F., Andre, D., and Keane, M. (1999) *Genetic Programming III: Darwinian Invention and Problem Solving,* Morgan Kaufmann.

Maes, P. (1993). Behavior-based artificial intelligence. *From Animals to Animats: Proceedings of the Second International Conference on Simulation of Adaptive Behavior.* MIT Press.

Filliat, D. and Meyer J. (2003). Map-based navigation in mobile robots. I. *A review of localization strategies, Journal of Cognitive Systems Research*. (in press).

Meyer, J., and Filliat, D. (2003). Map-based navigation in mobile robots. II. A review of map-learning and path-planning strategies. Journal of Cognitive Systems Research. (in press).

Moravec, H.P. (1984). Locomotion, vision, and intelligence. *Robotics Research 1*. 215-224. MIT Press.

Pryor, R.J. (1998). Developing robotic behavior using a genetic programming model. SAND98-0072, Sandia National Laboratories.

Pryor, R.J. & Barton, (2002). Developing maneuvering behaviors for a glider UAV using a genetic programming model. SAND2002-3147, Sandia National Laboratories.

Sharkey, N. and Ziemke, T. (2000). Life, mind and robots; the ins and outs of embodied cognition. *Hybrid Neural Systems*. Heidelberg: Springer Verlag.

Sims, K. (1994). Evolving 3D morphology and behavior by competition. *Artificial Live IV*. Proceedings, MIT Press.

Weiss, G. (1999). *Multiagent Systems.* MIT Press.

Wilson, S. W. (1985). Knowledge growth in an artificial animal. *Proceedings of the First International Conference on Genetic Algorithms and their Applications*. Lawrence Erlbaum Associates.

Wooldridge, M. and Jennings, N. (1994). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*.